Section I:

AMENDMENT UNDER 37 CFR §1.121 to the CLAIMS

Claim 1 (currently amended):

A method of debugging an application comprising Java code and native method dynamic load libraries, the method operating in a computer having an operating system, a system debug application programming interface (API), and a Java Virtual Machine having a Java Platform Debugger Architecture (JPDA) Virtual Machine Debug Interface API, said method comprising the steps of:

launching a Java virtual machine (JVM) under the system debug API;

using a JPDA bootstrap object to obtain a JPDA Virtual Machine Manager;

accessing a list of connectors from said Virtual Machine Manager;

connecting a probe program to said JVM by iterating through said list of connectors and storing a command line launch object as a connector object;

executing [[the]] an application program for debug and test under the ~~Java virtual machine~~ JVM by setting a main argument for the connector object to the name of said application program, setting a suspend argument for the connector object to true, invoking a launch method, which returns a Virtual Machine (VM) object to the probe program, creating one or more exception requests for suspending the application program when an exception event occurs, creating a notification request for loading of classes and thread deaths, reading a list of loaded classes, and spawning a thread which enters an event retrieval loop; and

simultaneously debugging [[the]] Java code of said application program and [[the]] one or more native method dynamic load libraries as the application is executing under the Java virtual machine, wherein the Java code is debugged using the Java Platform Debugger Architecture Virtual Machine Debug Interface API and wherein the native method dynamic load libraries are debugged using the system debug API.

Claim 2 (original):

The method as described in Claim 1 wherein the step of simultaneously debugging the Java code and the native method dynamic load libraries includes providing and displaying a console message window which permits a user to view output printed by the application.

Claim 3 (original):

The method as described in Claim 2 wherein said step of providing and displaying a console message window includes the steps of:

obtaining a Process object from the virtual machine;

obtaining an InputStream object from the Process object,

creating a "BufferedReader" object; and

using a read method to retrieve data from the application.

Claim 4 (original):

The method as described in Claim 1 wherein the step of simultaneously debugging the Java code and the native method dynamic load libraries includes providing and displaying a console message window which permits a user to write data to the application.

Claim 5 (original):

The method as described in Claim 4 wherein said step of providing and displaying a console message window includes the steps of:

obtaining a Process object from the virtual machine;

obtaining an OutputStream object from the Process object,

creating a "BufferedWriter" object; and

using a write method to write data to the application.

Claim 6 (original):

The method as described in Claim 1 wherein the step of simultaneously debugging the Java code and the native method dynamic load libraries includes providing the ability to exclude classes when stepping execution by use of an asterisk when invoking an "addClassExclusionFilter" method on a step request object .

Claim 7 (original):

The method as described in Claim 1 wherein the step of simultaneously debugging the Java code and the native method dynamic load libraries includes providing string evaluation with a StringReference interface which extends an ObjectReference such that a string is treated as a class.

Claim 8 (original):

The method as described in Claim 1 wherein the step of simultaneously debugging the Java code and the native method dynamic load libraries includes modifying symbols such as static and instance fields, strings, array elements, and Booleans, by:

invoking a mirrorOf utility to obtain an access reference to a type of symbol to be modified; and

invoking a setValue utility to modify the value of the symbol.

Claim 9 (original):

The method as described in Claim 1 wherein the step of simultaneously debugging the Java code and the native method dynamic load libraries includes setting a breakpoint by:

obtaining a list of the loaded classes;

obtaining line number information from a class object;

instantiating a breakpoint request;

setting the breakpoint request suspend policy; and

enabling the breakpoint request.

Claim 10 (original):

The method as described in Claim 9 wherein the step of setting a breakpoint further comprises adding the breakpoint to a deferred breakpoint table if the class where the breakpoint is to be located is not already loaded.

Claim 11 (original):

The method as described in Claim 1 wherein the step of simultaneously debugging the Java code and the native method dynamic load libraries includes loading classes by:

obtaining a reference to a method "getSystemClassLoader"; and

invoking said method to load a class.

Claim 12 (original):

The method as described in Claim 1 wherein the step of simultaneously debugging the Java code and the native method dynamic load libraries includes stepping through execution of the application by:

obtaining an event request manager from the virtual machine;

invoking a method of the event request manager to create a new step request including a step size parameter;

enabling the new step request; and

resuming the virtual machine.

Claim 13 (original):

The method as described in Claim 1 wherein the step of simultaneously debugging the Java code and the native method dynamic load libraries includes handling events from the virtual machine by:

providing a plurality of event handlers, one event handler for each requested event object to be processed;

polling a virtual machine event queue to determine if any requested event objects are queued;

removing requested event objects from the event queue;

determining the event object type; and

calling an event handler according to the determined event object type.

Claim 14 (currently amended):

A computer-readable medium containing program code for a software development workstation for debugging an application comprising Java code and native method dynamic load libraries, the software development workstation comprising a computer with an operating system, a system debug application programming interface (API), and a Java Virtual Machine having a Java Platform Debugger Architecture Virtual Machine Debug Interface API (JPDA), said program code when executed by said computer causing said computer to perform the steps of:

launching a Java virtual machine (JVM) under the system debug API;

using a JPDA bootstrap object to obtain a JPDA Virtual Machine Manager;

accessing a list of connectors from said Virtual Machine Manager;

connecting a probe program to said JVM by iterating through said list of connectors and storing a command line launch object as a connector object;

executing [[the]] an application program for debug and test under the ~~Java virtual machine~~ JVM by setting a main argument for the connector object to the name of said application program, setting a suspend argument for the connector object to true, invoking a launch method, which returns a Virtual Machine (VM) object to the probe program, creating one or more exception requests for suspending the application program when an exception event occurs, creating a notification request for loading of classes and thread deaths, reading a list of loaded classes, and spawning a thread which enters an event retrieval loop; and

simultaneously debugging [[the]] Java code of said application program and [[the]] one or more native method dynamic load libraries as the application is executing under the Java virtual machine, wherein the Java code is debugged using the Java Platform Debugger Architecture Virtual Machine Debug Interface API and wherein the native method dynamic load libraries are debugged using the system debug API.

Claim 15 (original):

The computer readable medium as described in Claim 14 wherein program code for performing the step of simultaneously debugging the Java code and the native method dynamic load libraries includes program code for providing and displaying a console message window which permits a user to view output printed by the application.

Claim 16 (original):

The computer readable medium as described in Claim 15 wherein program code for performing the step of providing and displaying a console message window includes program code for performing the steps of:

obtaining a Process object from the virtual machine;

obtaining an InputStream object from the Process object,

creating a "BufferedReader" object; and

using a read method to retrieve data from the application.

Claim 17 (original):

The computer readable medium as described in Claim 14 wherein program code for performing the step of simultaneously debugging the Java code and the native method dynamic load libraries includes program code for providing and displaying a console message window which permits a user to write data to the application.

Claim 18 (original):

The computer readable medium as described in Claim 17 wherein said program code for performing the step of providing and displaying a console message window includes program code for performing the steps of:

obtaining a Process object from the virtual machine;

obtaining an OutputStream object from the Process object,

creating a "BufferedWriter" object; and

using a write method to write data to the application.

Claim 19 (original):

The computer readable medium as described in Claim 14 wherein program code for performing the step of simultaneously debugging the Java code and the native method dynamic load libraries includes program code for providing the ability to exclude classes when stepping execution by use of an asterisk when invoking an addClassExclusionFilter method on a step request object .

Claim 20 (original):

The computer readable medium as described in Claim 14 wherein program code for performing the step of simultaneously debugging the Java code and the native method dynamic load libraries includes program code for string evaluation with a StringReference interface which extends an ObjectReference such that a string is treated as a class.

Claim 21 (original):

The computer readable medium as described in Claim 14 wherein program code for performing the step of simultaneously debugging the Java code and the native method dynamic load libraries includes program code for modifying symbols such as static and instance fields, strings, array elements, and Booleans, by:

invoking a mirrorOf utility to obtain an access reference to a type of symbol to be modified; and

invoking a setValue utility to modify the value of the symbol.

Claim 22 (original):

The computer readable medium as described in Claim 14 wherein program code for performing the step of simultaneously debugging the Java code and the native method dynamic load libraries includes program code for setting a breakpoint by:

obtaining a list of the loaded classes;

obtaining line number information from a class object;

instantiating a breakpoint request (107);

setting the breakpoint request suspend policy; and

enabling the breakpoint request.

Claim 23 (original):

The computer readable medium as described in Claim 22 wherein program code for performing the step of setting a breakpoint further comprises program code for adding the breakpoint to a deferred breakpoint table if the class where the breakpoint is to be located is not already loaded.

Claim 24 (original):

The computer readable medium as described in Claim 14 wherein program code for performing the step of simultaneously debugging the Java code and the native method dynamic load libraries includes program code for loading classes by:

obtaining a reference to a method "getSystemClassLoader"; and

invoking said method to load a class.

Claim 25 (original):

The computer readable medium as described in Claim 14 wherein program code for performing the step of simultaneously debugging the Java code and the native method dynamic load libraries includes program code for stepping through execution of the application by:

obtaining an event request manager from the virtual machine;

invoking a method of the event request manager to create a new step request including a step size parameter;

enabling the new step request; and

resuming the virtual machine.

Claim 26 (original):

The computer readable medium as described in Claim 14 wherein program code for performing the step of simultaneously debugging the Java code and the native method dynamic load libraries includes program code for handling events from the virtual machine by:

providing a plurality of event handlers, one event handler for each requested event object to be processed;

polling a virtual machine event queue to determine if any requested event objects are queued;

removing requested event objects from the event queue;

determining the event object type; and

calling an event handler according to the determined event object type.

Claim 27 (currently amended):

A computer, comprising:

a processor;

an operating system having a system debug application programming interface (API);

a Java interpreter having an associated Java Platform Debugger Architecture API for running an application program under a Java Virtual Machine (JVM);

a Java probe means for accessing and controlling an application program being executed by said Java Virtual Machine via said Java Platform Debugger Architecture API; and

a debugger engine daemon, communicative to said Java probe, for simultaneously debugging the Java code and native method dynamic load library code in coordination with said probe means, said simultaneous debugging being performed by using a JPDA bootstrap object to obtain a JPDA Virtual Machine Manager, accessing a list of connectors from said Virtual Machine Manager, connecting a probe program to said JVM by iterating through said list of connectors and storing a command line launch object as a connector object, executing anapplication program for debug and test under the JVM by setting a main argument for the connector object to the name of said application program, setting a suspend argument for the connector object to true, invoking a launch method, which returns a Virtual Machine (VM) object to the probe program, creating one or more exception requests for suspending the application program when an exception event occurs, creating a notification request for loading of classes and thread deaths, reading a list of loaded classes, and spawning a thread which enters an event retrieval loop, and simultaneously debugging Java code of said application program and one or more native method dynamic load libraries as the application is executing under the Java virtual machine.

Claim 28 (original):

The computer as described in claim 27 wherein the debugger engine daemon is adapted to debug native method dynamic load libraries of the application under the system debug API and the Java probe is adapted to debug Java application code under the Java Platform Debugger Architecture.

Claim 29 (original):

The computer as described in claim 27 further comprising a console message window means capable of reading data from the application and displaying it to a user, said console message window means comprising:

a Process object obtained from the virtual machine;

an InputStream object obtained from the Process object,

a "BufferedReader" object; and

a read method adapted to retrieve data from the application and display the retrieved data on a console message window display.

Claim 30 (original):

The computer as described in claim 27 further comprising a console message window means capable of writing data to the application under user control, said console message window means comprising:

a Process object obtained from the virtual machine;

an OutputStream object obtained from the Process object,

a "BufferedWriter" object; and

a write method adapted to write data to the application from a console message window display.

Claim 31 (original):

The computer as described in claim 27 further comprising a class excluder adapted to exclude classes when stepping program execution by use of an asterisk when invoking an "addClassExclusionFilter" method on a step request object.

Claim 32 (original):

The computer as described in claim 27 further comprising a string evaluator with a StringReference interface which extends an ObjectReference such that a string is treatable as a class.

Claim 33 (original):

The computer as described in claim 27 further comprising a symbol modifier for modifying symbols such as static and instance fields, strings, array elements, and Booleans, comprising:

a mirrorOf utility invoker adapted to obtain an access reference to a type of symbol to be modified; and

a setValue utility invoker adapted to modify the value of the symbol.

Claim 34 (original):

The computer as described in claim 27 further comprising a breakpoint setter comprising:

a list of the loaded classes obtained from the virtual machine;

line number information obtained from a class object;

a instantiated breakpoint request;

a breakpoint request suspend policy; and

a breakpoint request enabler.

Claim 35 (original):

The computer as described in claim 34 wherein said breakpoint setter further comprises a deferred breakpoint table for storing breakpoints for classes which are not currently loaded.

Claim 36 (original):

The computer as described in claim 27 further comprising an execution stepper comprising:

an event request manager obtained from the virtual machine;

a new step request method invoker for invoking a method of the event request manager to create a new step request including a step size parameter;

a new step request enabler; and

a virtual machine resumer.

Claim 37 (original):

The computer as described in claim 27 further comprising an event handler for handling events from the virtual machine comprising:

a plurality of event type handlers, one event type handler for each requested event object type to be processed;

an event queue poller for polling the virtual machine to determine if any requested event objects are queued; and

an event type handler sorter for determining the event object type and calling an event type handler.